JIRAE, Vol. 3, No. 2, October 2018, 79–85 DOI: 10.9744/JIRAE.3.2.79-85
e-ISSN 2407-7259

Game Popularity Tracking System

Joseph Alexander Budiarto

Department of Informatics, Petra Christian University, Surabaya, Indonesia
Information and Communication Technology, Fontys University of Applied Sciences, Eindhoven, The Netherlands
budiartojoseph@gmail.com

Abstract. The purpose of this research is to develop a game popularity tracking system that can generate popularity values once every 24 hours using daily data from Google Analytics. Within the scope of this project, game popularity is defined as the fact that the quality of a certain game is liked by many people. There are 5 metrics that can be used to calculate game popularity: users count, unique page views (UPV), average time on page (ATP), the difference of unique page views, and average time on page from the day before. To generate the popularity values daily, the combination of weighted average, exponential moving average, and exponential smoothing algorithm are used. The final result of this project is a service that can run automatically and also provides an HTTP API that could fetch popularity values of games from gaming sites, sending the data in XML format for other services to use when needed. The service is built using Node.JS and other in-trend technologies such as Google Analytics API v4, OAuth 2.0, Express.JS, etc.

Keywords: Game Popularity, Popularity Algorithm, Google Analytics, Node.JS, HTTP API

1. Introduction

Over the centuries, technologies have grown rapidly. Perhaps one of the greatest technological discoveries is the Internet. In the world today, the Internet has become one of the biggest platforms for people to interact with one another. Moreover, almost everything relies on the Internet nowadays. The Internet is inevitable as there are many things that people can find across the Internet, starting from knowledge, businesses, entertainment and much more. Entertainment on the Internet has strong relations with advertising. The common practice is that businesses will earn revenue from ads placed on their websites. In order to do that, they need a noticeable amount of traffic, which will attract advertising companies to place ads on their websites. The main purpose of making a popularity tracking system is to offer popular games for users to make it more comfortable for users to find which game is liked the most within a certain gaming site. The main task of this project is to propose and create a new popularity tracking system that can update a game popularity value automatically every day.

2. Research Results

The research phase of this project is divided into several stages. The first stage is to look into what the definition of 'game popularity' really is. After the definition is known, the research phase can continue to figure out what metrics to use, which data to collect in relation to the chosen metrics, and what algorithms to use for this project.

2.1. Game Popularity Definition

Popularity, as defined in Cambridge Dictionary, is the fact that someone or something is liked and/or supported by a lot of other people [1]. Gananath and Sreenath [2] described popularity as the quality of being liked or accepted by people. Based on these two definitions, game popularity can be defined as the fact that the quality of a certain game is liked by many people, which means it is safe to assume that *popularity has something to do with the users and/or customers*.

2.2. Metrics Used for the Calculation

According to Trevor McCalmont [3], there are 15 metrics that can be used to measure the quality of a game. He divided the metrics into 3 categories: basic, monetization, and in-game metrics (shown in Table 1).

Table 1. General game metrics

Categories	Metrics
Basic	Daily Active Users (DAU)
	Monthly Active Users (MAU)
	Sessions
	Retention Rate
	Churn Rate
	Ratio of DAU to MAU
Monetization	Conversion Rate
	Avg. Revenue per Daily Active User (ARPDAU)
	Avg. Revenue per Paying User (ARPPU)
In-Game	Source
	Sink
	Flow
	Start
	Fail
	Complete

The metrics defined by Trevor McCalmont that correspond to the popularity of a game are the basic metrics. Other than that, according to Gaurav Bagur [4], the number of users giving ratings and the actual value of those ratings itself can be used to measure the popularity value of a game.

The metrics in Google Analytics are clustered into 33 categories: User, Session, AdWords, Page Tracking, etc [5]. The category that can be used for this project is the Page Tracking category. The metrics available under the Page Tracking category are Page Value, Page Views, Unique Page Views, Average Time on Page, etc. The rest can be seen in Figure 1. Other than these metrics, Average Session Duration and Bounce Rate are also taken into consideration.

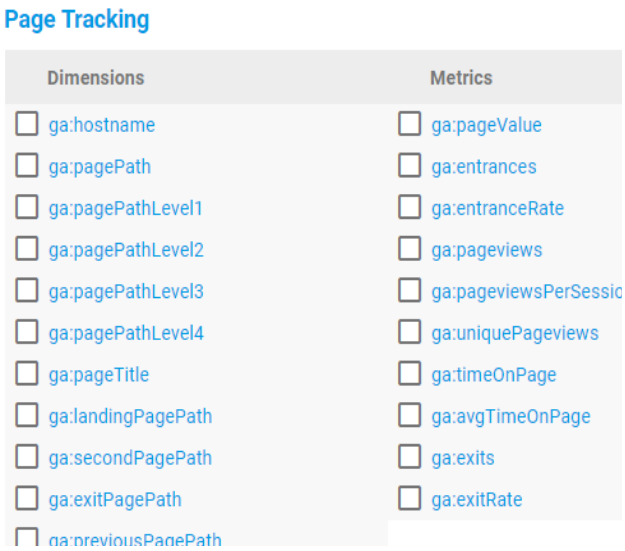

Figure 1. Google Analytics dimensions and Metrics [5]

To summarize the findings from the previous steps, the general game metrics to be used are: DAU, MAU, Session, Ratio of DAU to MAU, Retention Rate, Churn Rate, and lastly User Ratings. The User Rating used in this project consists of the accumulation of counts and stars given by users. The metrics of Google Analytics most likely to be used are the ones under Page Tracking category, Average Session Duration, and Bounce rate. When relating the two findings, it can be assumed as such: Daily Active Users (DAU) corresponds to Unique Page Views, Sessions corresponds to Average Time on Page (ATP) or Average Session Duration, and Churn Rate corresponds to Bounce Rate. Out of the four Google Analytics metrics mentioned, Average Session Duration and Bounce Rate *CANNOT* be used to calculate the popularity score, because these two metrics are not bound to only one web page. There are 2 other metrics that can also be used to calculate the daily popularity value. When looking into daily changes of popularity, the difference between today's and yesterday's data is also important. For example, if a game has a huge positive difference of user playing today compared to the previous day, then we can assume that the game is popular today. To conclude everything, here are the metrics that will be used to calculate the new popularity value of games:

1. User Rating Count
2. Unique Page Views (Google Analytics Metrics)
3. Average Time on Page (Google Analytics Metrics)
4. Difference of Unique Page Views (Diff UPV)
5. Difference of Average Time on Page (Diff ATP)

2.3. Data Validity

The data acquired from Google Analytics are *NOT* 100% valid. According to Ryan Chase [6], there are several reasons as to why Google Analytics data are not 100% accurate, and never will be. Below are the three main factors that can affect Google Analytics data validity:

1. Implementation error
   Implementation error can be in the form of including the script twice in one page or other errors. However, the implementation of Google Analytics is not part of this project and will not be discussed further.
2. Google Analytics' sample data
   When Google Analytics does not have enough data, it uses sample data to calculate the value of certain metrics. Even though the accuracy of those sample data can be set manually, it is still not real data which is why the result will never be valid.
3. Disabled JavaScript & cookies, and ad-blocks
   The third reason is because some users that visit the sites sometimes disable their JavaScript and/or cookies, and sometimes also uses ad-blocks. This causes Google Analytics to fail when trying to track the data from users. According to Matthew Cortland [7], in 2017 11% of users worldwide actually use ad-blocks. In 2016, Andrzej Winnicki [8] from Yell – UK's leading online business directory – conducted a research and found out that only 0.07% of users from Yell.com disabled their JavaScript and 0.2% didn't allow cookies.

In conclusion, after seeing all the facts above, the data acquired from Google Analytics is *NOT* 100% valid because some of the factors are outside of this project's scope and control.

2.4. Algorithm Research

There are 12 algorithms in total that came up after searching for several references on how to calculate popularity of a game from the Internet: Those are the Wilson Score Interval, the Elo Rating System, Rank Product, Exponential Moving Average, Weighted Average, Weighted Moving Average, Exponential Weighted Moving Average, Bayesian Average, Normalization, Euler's Number, Exponential Smoothing, and Cross Multiplication. It is important to note that not all the algorithms above can be used to calculate a popularity value. Some of them are excellent

for generating rating scores, some can be used to generate the game rankings, etc. The final algorithm used to calculate the popularity value is a combination between Weighted Average (WA) and Exponential Moving Average (EMA), and Exponential Smoothing to scale the EMA result so it ranges from 0 – 100.

2.4.1. Weighted Average

$$WA = \frac{\sum_{i=1}^{n} wi.xi}{\sum_{i=1}^{n} wi} \quad (1)$$

The way the weighted average works is by multiplying the data from one metric with its preset weight. Then, all the multiplication result is summed and divided by the total weight. For an example: game A has *X* and *Y* as metrics. The data and weights for each metric are as follows: $W_X = 3$; $X = 5$; $W_Y = 1$; $Y = 7$. The weighted average of Game A can be calculated as:

$$WA = \frac{\big((W_X \times X) + (W_Y \times Y)\big)}{(W_X + W_Y)} = \frac{(15 + 7)}{4} = 5.5$$

2.4.2. Exponential Moving Average

$$EMA_{today} = EMA_{yesterday} + \alpha\,[WA_{today} - EMA_{yesterday}] \quad (2)$$

$$\alpha = (2 \div (N + 1)) \quad (3)$$

After the weighted average has been calculated, the time factor is taken into account. The bigger the N, the smaller the data difference will be, and vice versa. For an example: continuing from the previous example, if Game A has yesterday's EMA value $EMA_{yesterday} = 8$, and *N* is set to 28, then the next step of calculation is:

$$EMA_{today} = 8 + \big(2 \div (28 + 1)\big) \times [5.5 - 8] = 7.83$$

2.4.3. Exponential Smoothing (Modified)

$$s_0 = .\,x_0 + (1 - \alpha)\,.\,1000 \quad (4)$$

$$s_t = \alpha\,.\,x_t + (1 - \alpha)\,.\,_{t-1} \quad (5)$$

(where $0 < \alpha < 1$)

The result of this calculation is not used for the next day popularity calculation. The sole purpose of this algorithm is just to smooth the popularity value so it is presentable for the users. For an example: previously, Game A's EMA value for today is 7.83. If Game B's EMA value for today is 8, then Game B will be placed above Game A. Thus, Game B's EMA value will be used as s_0, and Game A will be s_1. If the α is set to 0.0005, the calculation process is:

$$s_0 = 0.0005 \times 8 + (1 - 0.0005) \times 1000 = 999.5 \approx 1000$$

$$s_1 = 0.0005 \times 7.83 + (1 - 0.0005) \times 999.5 = 999$$

To make it range from 0 – 100, the result will then be divided by 10. By doing so, Game A's popularity value is 100 and Game B's popularity value is 99.9. The smoothing results before divided by 10 are the ones used for the next game smoothing calculation, while the division-by-10 results are used to present the popularity value for the users. It is important to note that there are several constraints when using this algorithm:

1. All metrics used for the calculation have to be normalized.
2. When the algorithm is used for the first time, the weighted average result is set as the initial popularity value; i.e., $EMA_{initial} = WA_{today}$.
3. There is no default popularity value for a newly published game.
4. There is a User Rating Limit when calculating the Weighted Average.

2.5. Algorithm Customization

One of the advantages of using Weighted Average to calculate the popularity value is that by setting the weights differently, the calculation results will vary, depending on how much weight is assigned to each metric. The damping factor $(1 - \alpha)$ of Exponential Moving Average algorithm can also be customized. The larger the damping factor, the less varied the results will be, and vice versa. These settings were made so that the algorithm can be tailored and produce different sets of results. It can also be useful for future development, for example, adding or removing metrics used in the algorithm. Three predetermined settings that can be used to get different sets of results are shown in Tables 2 to 4.

Table 2. Set 1

No.	Metrics	Weight
1.	Users Count	3
2.	UPV	5
3.	Difference of UPV	0
4.	ATP	1
5.	Difference of ATP	0
6.	*N*	28

Table 3. Set 2

No.	Metrics	Weight
1.	Users Count	3
2.	UPV	5
3.	Difference of UPV	1
4.	ATP	1
5.	Difference of ATP	1
6.	*N*	28

Table 4. Set 3

No.	Metrics	Weight
1.	Users Count	3
2.	UPV	5
3.	Difference of UPV	2
4.	ATP	2
5.	Difference of ATP	1
6.	*N*	2

2.6. Decisions

After looking into each of the algorithms and its endless possibilities, in the end, some decisions need to be made. Below can be found the summarized decision points of the algorithm research phase:

1. Before calculating the Weighted Average, the data are normalized.
2. The weight of each metric can be customized.
3. Exponential Moving Average deals with the time factor when generating the new popularity value.
4. Exponential Smoothing will scale the EMA results so it ranges from 0 – 100 with 1 fraction digit.
5. The EMA_{today} value is the one being used for the next day's popularity calculation.
6. The algorithm is designed to have minimal change, so initial popularity value can be customized and, in the end, it affects the final calculation results.

3. Implementation Results

Figure 2 shows how the core functions of this new system work to generate all games' popularity values.

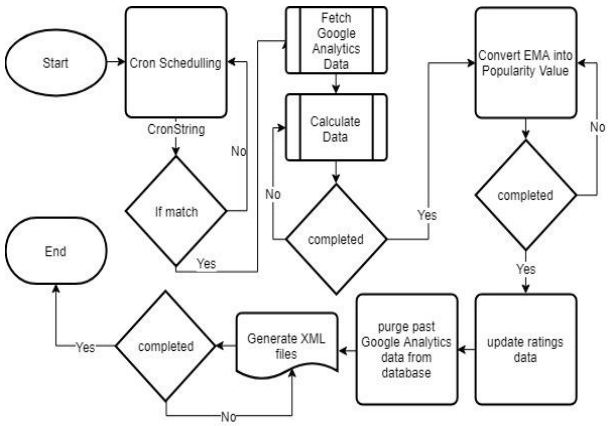

Figure 2. Core functions of the new system

For the implementation, JavaScript programming language is used (Node.JS). The database chosen is MySQL. In order to help manage the program, PM2 is also implemented as the process manager. To get data from Google Analytics, Google Analytics API (Node.JS library) is used.

3.1. Google Analytics API

There are several steps to take when preparing to call for Google Analytics API. To actually get data from Google Analytics, a user needs to be authenticated first. There are several ways to authenticate a user and receive an access token, two of which are using OAuth 2.0 client IDs or using service account keys [9]. After the user is authenticated, the next step is to call the API and provide the query parameters needed to fetch the desired data. There are several metrics and dimensions that the user can choose from the API.

When sending the query request to Google Analytics API, start-date and end-date parameters need to be specified; in this case, the start-date and the end-date parameters are set to yesterday's date. By doing so, when the program runs automatically at the beginning of the day, the data gathered from Google Analytics will be a complete set from the day before.

3.2. Algorithm Implementation

The algorithm used in this project is implemented and can be seen from the core functions. These core functions are called when the cron scheduling is executed. It consists of functions for fetching, calculating, scaling, updating, purging of data, and generating XML files containing popularity values of games from all sites.

3.2.1. Fetching Data

This function is the very first function to run when the application is executed: fetching data from Google Analytics and storing it in the database. During this process, the Google Analytics API is used. This API needs a service account key for authentication. After loading the key, then use the code as seen in Figure 3 to call the API.

```
jwtClient.authorize((err, tokens) => {
    if (err) {
        console.error(err);
        return;
    }
    let analytics = google.analytics('v3');
    analytics.data.ga.get({
        'auth': jwtClient,
        'ids': 'ga:xxxxxxxx',
        'metrics': 'ga:uniquePageviews,ga:avgTimeOnPage',
        'dimensions': 'ga:pagePath',
        'start-date': start || Popularity.config.start_date,
        'end-date': end || Popularity.config.end_date,
        'sort': '-ga:uniquePageviews',
        'max-results': 10000,
    }, (err, response) => {
        if (err) {
            reject(err);
        } else {
            resolve(JSON.stringify(response.data.rows));
        }
    });
});
```

Figure 3. Google Analytics API code snippet

One important remark is that the *data gathered from Google Analytics does not represent all games available on one gaming site*. Games that are not visited by users will not appear in Google Analytics. That being said, this function also needs a logic to combine games that have today's data with the ones that do not. For example, game A has some UPV for today, and on the other hand, game B does not. In cases like this, game A will appear on Google Analytics, but game B will not. However, the calculation of game popularity will also take game B into account when calculating the popularity value for today. If game B does not have any data for today, then that means game B's popularity is decreasing.

3.2.2. Calculating Data

During this phase, all data from Google Analytics are already available in the database. This is to prevent any error caused by unavailable data because Node.JS

can run asynchronously, meaning that it does not have to wait until one process is finished to run the next one. The algorithm devised during the research phase is implemented here.

1. Calculate the difference metrics.
   To calculate the difference of UPV and the difference of ATP, previous day's data is needed. This data can be found by querying to analytics data using the date of the previous day. If there are no data from the previous day then both of the metrics will be set to zero. The data will be temporarily saved in an array.
2. Merge data of all games.
   After the differences of UPV and ATP are calculated, this function will query all games available in respective gaming sites. After that, the data from Google Analytics will be merged together with the result from the query, so by now the process already has all game data available.
3. Get game ratings for the current day.
   While the process is merging data, it will also send queries to the database to get game ratings, consisting of the users count and the stars given by those users. It will then subtract the values of users count and stars today with the values of old users count and old stars from the column old count and old stars. All the ratings data will also be temporarily held in an array. (**Note:** Steps 2 and 3 run concurrently and both functions are asynchronous.)
4. Calculate Weighted Average.
   After all metrics have been gathered, the next step is calculating the weighted average. By using the weight set for production, the algorithm for Weighted Average is implemented here. Also, the limit of user ratings is implemented. After all calculations are done, then the process can move on to calculating the popularity value, which considers the previous popularity value using Exponential Moving Average.
5. Calculate Exponential Moving Average.
   Before calculating the popularity value, this function will execute a query to the database to get the EMA value of prior day for each game. If there is no previous data, then the weighted average is set as the initial value (see subsection 2.4.3, the second constraint). If there is data from the previous day, the EMA algorithm is executed.

3.2.3. Scale EMA to Generate Popularity Value

This step is only for readability purposes. The EMA result is smoothed using exponential smoothing to make it range from 0 – 100. The results will represent the popularity. However, for the next day's calculation, the EMA value stored in the database is used instead.

3.2.4. Update Ratings Data

This function is executed after all calculation processes have finished. The main usage of this function is to update the value in the database. Because the user ratings consist of the accumulation of users count and stars, there are two extra fields that need to be updated after every calculation. These two fields record the old values of users count and stars. When doing the calculation, these two fields are used to get the users count and stars of one day only, by subtracting the accumulation of users count and stars with the data from these two fields.

$$count_{today} = count_{accumulation} - count_{old} \quad (6)$$

$$stars_{today} = stars_{accumulation} - stars_{old} \quad (7)$$

3.2.5. Purge Data

The purge function is used for deleting old data from the database. The reason is that the Google Analytics data from previous days are no longer used for future popularity calculation. Thus, the data are purged in order to free up some space in the database.

3.2.6. Generating and Saving XML Files

The last part of the core function is to read from the popularity and game rating table in the database, then generate the XML files containing all games data available in the database. These files will be used by the API to retrieve the desired popularity values.

3.3. Application Programming Interface

This project is using RESTful API to get the popularity value of games. The standard way to access the API is by sending a GET request to the specified URL. The other HTTP methods are not used because the sole purpose of this API is to read data, not create nor update. The way the API works is receiving the request with parameters set by the client, and then using the parameters to read the desired XML file. The API will then send a response, back to the client, containing all the content of the XML file.

The request can be sent by either using HTTP or HTTPS, and it will return data in XML format, containing the popularity and the ratings of games.

3.4. Scheduling

To keep the application running, it is deployed in a virtual machine and managed by the help of PM2. PM2 is an advanced Node.JS process manager [10]. Even if the application crashed, PM2 can automatically restart the application and write the errors into a log file. To automatically run the functions to calculate the popularity value, cron scheduling is used. E.g., by specifying the cron string to '0 0 8 * * *', the calculation will be executed every day at 8 am. By

combining the Google Analytics service account authentication, PM2, and the cron scheduling, the application can run automatically.

3.5. Implementation Remarks

Based on the implementation results in the previous chapter, here are some important remarks about this project:

1. Game Popularity definition can be defined as the fact that the quality of a certain game is liked by many people.
2. Metrics that represent game popularity, based on the prior definition of game popularity, are the number of users giving ratings (users count), unique page views, and average time on page.
3. Google Analytics can be used to gather daily data to calculate popularity values, even though the data validity is *NOT* 100%.
4. The combination of Weighted Average, Exponential Moving Average, and Exponential Smoothing can produce the desired popularity value.
5. Exponential Moving Average can be used to deal with time factor when calculating the game popularity values.
6. Cron Scheduling can be used to make the program run automatically and PM2 can be used to keep the program running continuously.

4. Testing

A/B testing method is used to test the formula. In order to do this, real production data is needed. However, unfortunately, due to *CONFIDENTIALITY* reasons, the data retrieved from the company is limited, and the name of the company cannot be mentioned. The games will also be referred to as games A, B, C, D, and E. The test data can be seen in Table 5 and it was obtained on August 9, 2018, from the company's gaming website (*the source will not be mentioned*). The popularity value shown in the table was calculated using the *company's formula*, and it has been around for more than 2 years, meaning that the formula has been proven to work in the production environment.

Table 5. Production's popularity data

Game	Popularity Value
A	4,573.395
B	2,446.182
C	1,477.791
D	715.576
E	646.369

Table 6 shows the calculation results of game A – E, also obtained on August 9, 2018. Unfortunately, the initial popularity value is unavailable, so according to the formula's second constraint (see subsection 2.4.3), the weighted average results will be set as the initial popularity values.

Table 6. Calculation results

Game	UPV	ATP	Count	WA
A	2,561	5,037.42	5,751	3,899.49
B	1,042	2,052.92	3,137	1,852.66
C	561	2,221.38	1,457	1,044.15
D	311	2,518.84	581	646.32
E	195	2,900.58	559	616.95

In table 6, the formula only considers the UPV, ATP, and Count, and not the difference of UPV and difference of ATP. That is because no previous data is available. The parameter setting used in the calculation can be seen in Table 2. Table 7 shows the comparison between the company's formula and the new popularity formula.

Table 7. Comparison of popularity results

Game	Company's Formula	New Formula
A	4,573.395	3,899.49
B	2,446.182	1,852.66
C	1,477.791	1,044.15
D	715.576	646.32
E	646.369	616.95

Even though the numbers are clearly different, if the games are sorted, then the game list will have the same order. The main difference of the old and the new formula is that *the new formula takes into consideration several metrics that affect the popularity of a game*, while on the other hand the old formula only considers the user given ratings of games. That being said, there are bound to be some differences between the old and the new formula over time, because the new formula also considers the time factor. Unfortunately, it is not possible to test it because the data is insufficient at the moment.

5. Conclusion

By combining several algorithms (normalization, weighted average, exponential moving average, and exponential smoothing), using the right metrics (unique page views, average time on page, and user ratings), and after testing it using A/B testing method, as can be seen from the result, the proposed formula can produce the popularity value of games. The weighted average algorithm makes it possible for the formula to be customized as needed, while the exponential moving average deals with the time factor.

After the popularity value is generated, the games can then be sorted based on those values, which will produce a list of games ranked by their popularity.

For implementation purposes, to make the program calculate all popularity values daily automatically, the use of PM2 and cron scheduling is more than sufficient. To access the result, an API that has the function to retrieve XML files from the server can be applied and accessed using the GET method only, because it only needs to read data from the server.

References

1. Cambridge Dictionary, *Popularity – Meaning in the Cambridge English Dictionary*, retrieved from https://dictionary.cambridge.org/dictionary/english/popularity.
2. Gananath, R. and Sreenath, R., *Calculating Popularity Using a Simple Algorithm*, Dec. 2013, retrieved from http://vixra.org/pdf/1312.0052v1.pdf.
3. McCalmont, T., *15 Metrics All Game Developers Should Know by Heart*, GameAnalytics, Jul. 2015, retrieved from https://gameanalytics.com/blog/metrics-all-game-developers-should-know.html.
4. Bagur, G., *How do you measure the popularity of games?*, Mar. 2015, retrieved from https://www.quora.com/How-do-you-measure-the-popularity-of-games.
5. Google Developers, *Dimensions & Metrics Explorer*, Analytics Reporting API v4, retrieved from https://developers.google.com/analytics/devguides/reporting/core/dimsmets#cats=session,user,page_tracking.
6. Chase, R., *Can You Trust Your Google Analytics Data?*, Blast Analytics & Marketing, Feb. 2013, retrieved from https://www.blastam.com/blog/can-you-trust-your-google-analytics-data.
7. Cortland, M., *2017 Adblock Report*, PageFair, Feb. 2017, retrieved from https://pagefair.com/blog/2017/adblockreport/.
8. Winnicki, A., *Just how many web users disable cookies or JavaScript?*, The Yell Blog, Apr. 2016, retrieved from https://blog.yell.com/2016/04/just-many-web-users-disable-cookies-javascript/.
9. Google Developers, *Analytics Reporting API - Authorization*, Analytics Reporting API v4, retrieved from https://developers.google.com/analytics/devguides/reporting/core/v4/authorization.
10. PM2, *PM2 - Advanced Node.js process manager*, retrieved from http://pm2.keymetrics.io/.